Network Working Group
Request for Comments: 983

D. E. Cass (NRTC)
M. T. Rose (NRTC)
April 1986

ISO Transport Services on Top of the TCP

Status of This Memo

This memo describes a proposed protocol standard for the ARPA Internet community. The intention is that hosts in the ARPA-Internet that choose to implement ISO TSAP services on top of the TCP be expected to adopt and implement this standard. Suggestions for improvement are encouraged. Distribution of this memo is unlimited.

1. Introduction and Philosophy

The ARPA Internet community has a well-developed, mature set of transport and internetwork protocols (TCP/IP), which are quite successful in offering network and transport services to end-users. The CCITT and the ISO have defined various session, presentation, and application recommendations which have been adopted by the international community and numerous vendors. To the largest extent possible, it is desirable to offer these higher level services directly in the ARPA Internet, without disrupting existing facilities. This permits users to develop expertise with ISO and CCITT applications which previously were not available in the ARPA Internet. It also permits a more graceful transition strategy from TCP/IP-based networks to ISO-based networks in the medium- and long-term.

There are two basic approaches which can be taken when "porting" an ISO or CCITT application to a TCP/IP environment. One approach is to port each individual application separately, developing local protocols on top of the TCP. Although this is useful in the short-term (since special-purpose interfaces to the TCP can be developed quickly), it lacks generality.

A second approach is based on the observation that both the ARPA Internet protocol suite and the ISO protocol suite are both layered systems (though the former uses layering from a more pragmatic perspective). A key aspect of the layering principle is that of layer-independence. Although this section is redundant for most readers, a slight bit of background material is necessary to introduce this concept.

Externally, a layer is defined by two definitions:

> a service-offered definition, which describes the services provided by the layer and the interfaces it provides to access those services; and,

a service-required definitions, which describes the services used by the layer and the interfaces it uses to access those services.

Collectively, all of the entities in the network which co-operate to provide the service are known as the service-provider. Individually, each of these entities is known as a service-peer.

Internally, a layer is defined by one definition:

a protocol definition, which describes the rules which each service-peer uses when communicating with other service-peers.

Putting all this together, the service-provider uses the protocol and services from the layer below to offer the its service to the layer above. Protocol verification, for instance, deals with proving that this in fact happens (and is also a fertile field for many Ph.D. dissertations in computer science).

The concept of layer-independence quite simply is:

IF one preserves the services offered by the service-provider

THEN the service-user is completely naive with respect to the protocol which the service-peers use

For the purposes of this memo, we will use the layer-independence to define a Transport Service Access Point (TSAP) which appears to be identical to the services and interfaces offered by the ISO/CCITT TSAP (as defined in [ISO-8072]), but we will base the internals of this TSAP on TCP/IP (as defined in [RFC-793,RFC791]), not on the ISO/CCITT transport and network protocols. Hence, ISO/CCITT higher level layers (all session, presentation, and application entities) can operate fully without knowledge of the fact that they are running on a TCP/IP internetwork.

The authors hope that the preceding paragraph will not come as a shock to most readers. However, an ALARMING number of people seem to think that layering is just a way of cutting up a large problem into smaller ones, *simply* for the sake of cutting it up. Although layering tends to introduce modularity into an architecture, and modularity tends to introduce sanity into implementations (both conceptual and physical implementations), modularity, per se, is not the end goal. Flexibility IS.

2. Motivation

In migrating from the use of TCP/IP to the ISO protocols, there are several strategies that one might undertake. This memo was written with one particular strategy in mind.

The particular migration strategy which this memo uses is based on the notion of gatewaying between the TCP/IP and ISO protocol suites at the transport layer. There are two strong arguments for this approach:

a. Experience teaches us that it takes just as long to get good implementations of the lower level protocols as it takes to get good implementations of the higher level ones. In particular, it has been observed that there is still a lot of work being done at the ISO network and transport layers. As a result, implementations of protocols above these layers are not being aggressively pursued. Thus, something must be done "now" to provide a medium in which the higher level protocols can be developed. Since TCP/IP is mature, and essentially provides identical functionality, it is an ideal medium to support this development.

b. Implementation of gateways at the IP and ISO IP layers are probably not of general use in the long term. In effect, this would require each Internet host to support both TP4 and TCP. As such, a better strategy is to implement a graceful migration path from TCP/IP to ISO protocols for the ARPA Internet when the ISO protocols have matured sufficiently.

Both of these arguments indicate that gatewaying should occur at or above the transport layer service access point. Further, the first argument suggests that the best approach is to perform the gatewaying exactly AT the transport service access point to maximize the number of ISO layers which can be developed.

NOTE: This memo does not intend to act as a migration or intercept document. It is intended ONLY to meet the needs discussed above. However, it would not be unexpected that the protocol described in this memo might form part of an overall transition plan. The description of such a plan however is COMPLETELY beyond the scope of this memo.

Finally, in general, building gateways between other layers in the TCP/IP and ISO protocol suites is problematic, at best.

To summarize: the primary motivation for the standard described in

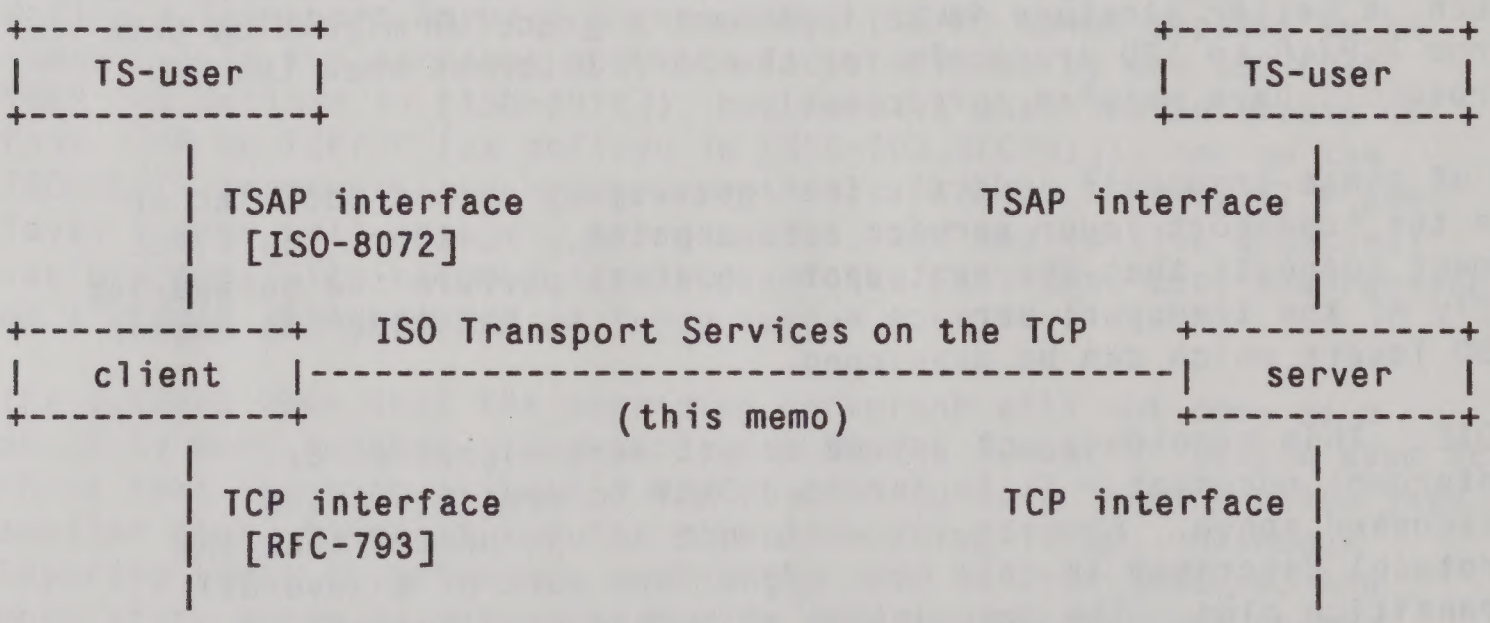
this memo is to facilitate the process of gaining experience with higher-level ISO protocols (session, presentation, and application). The stability and maturity of TCP/IP are ideal for providing solid transport services independent of actual implementation.

3. The Model

The [ISO-8072] standard describes the ISO transport service definition, henceforth called TP.

> ASIDE: This memo references the ISO specifications rather than the CCITT recommendations. The differences between these parallel standards are quite small, and can be ignored, with respect to this memo, without loss of generality. To provide the reader with the relationships:
>
> | | | |
> |---|---|---|
> | Transport service | [ISO-8072] | [X.214] |
> | Transport protocol | [ISO-8073] | [X.224] |
> | Session protocol | [ISO-8327] | [X.225] |

The ISO transport service definition describes the services offered by the TS-provider (transport service) and the interfaces used to access those services. This memo focuses on how the ARPA Transmission Control Protocol (TCP) [RFC-793] can be used to offer the services and provide the interfaces.



For expository purposes, the following abbreviations are used:

TS-peer	a process which implements the protocol described by this memo

TS-user	a process talking using the services of a TS-peer
TS-provider	the black-box entity implementing the protocol described by this memo

For the purposes of this memo, which describes version 1 of the TSAP protocol, all aspects of [ISO-8072] are supported with one exception:

Quality of Service parameters

In the spirit of CCITT, this is left "for further study". Version 2 of the TSAP protocol will most likely support the QOS parameters for TP by mapping these onto various TCP parameters.

Since TP supports the notion of a session port (termed a TSAP ID), but the list of reserved ISO TSAP IDs is not clearly defined at this time, this memo takes the philosophy of isolating the TCP port space from the TSAP ID space and uses a single TCP port. This memo reserves TCP port 102 for this purpose. This protocol manages its own TSAP ID space independent of the TCP. Appendix A of this memo lists reserved TSAP IDs for version 1 of this TSAP protocol. It is expected that future editions of the "Assigned Numbers" document [RFC-960] will contain updates to this list. (Interested readers are encouraged to read [ISO-8073] and try to figure out exactly what a TSAP ID is.)

Finally, the ISO TSAP is fundamentally symmetric in behavior. There is no underlying client/server model. Instead of a server listening on a well-known port, when a connection is established, the TS-provider generates an INDICATION event which, presumably the TS-user catches and acts upon. Although this might be implemented by having a server "listen" by hanging on the INDICATION event, from the perspective of the ISO TSAP, all TS-users just sit around in the IDLE state until they either generate a REQUEST or accept an INDICATION.

4. The Primitives

The protocol assumes that the TCP [RFC-793] offers the following service primitives:

Events

connected	- open succeeded (either ACTIVE or PASSIVE)
connect fails	- ACTIVE open failed
data ready	- data can be read from the connection
errored	- the connection has errored and is now closed
closed	- an orderly disconnection has started

Actions

listen on port	- PASSIVE open on the given port
open port	- ACTIVE open to the given port
read data	- data is read from the connection
send data	- data is sent on the connection
close	- the connection is closed (pending data is sent)

The protocol offers the following service primitives, as defined in [ISO-8072], to the TS-user:

Events

T-CONNECT.INDICATION

- a TS-user (server) is notified that connection establishment is in progress

T-DISCONNECT.INDICATION

- a TS-user is notified that the connection is closed

T-CONNECT.CONFIRMATION

- a TS-user (client) is notified that the connection has been established

T-DATA.INDICATION

- a TS-user is notified that data can be read from the connection

T-EXPEDITED DATA.INDICATION

- a TS-user is notified that "expedited" data can be read from the connection

Actions

T-CONNECT.RESPONSE

- a TS-user (server) indicates that it will honor the request

T-DISCONNECT.REQUEST

- a TS-user indicates that the connection is to be closed

T-CONNECT.REQUEST

- a TS-user (client) indicates that it wants to establish a connection

T-DATA.REQUEST

- a TS-user sends data

T-EXPEDITED DATA.REQUEST

- a TS-user sends "expedited" data

5. The Protocol

It is the goal of this memo to offer a TP interface on top of the TCP. Fortunately, the TCP does just about everything that TS-provider offers to the TS-user, so the hard parts of the transport layer (e.g., three-way handshakes, choice of ISS, windowing, multiplexing, ad infinitum) are all taken care of by the TCP.

Despite the symmetry of TP, it is useful to consider the protocol with the perspective of a client/server model.

The information exchanged between TSAP-peers is in the form of packets termed "TPKT"s. The format of these packets is described in the next section. For the purposes of the description below, a TPKT has a code which is one of:

CR - request connection
CC - confirm connection
DR - request disconnection
DT - data
ED - expedited data

A TSAP server begins by LISTENing on TCP port 102. When a TSAP client successfully connects to this port, the protocol begins.

A client decides to connect to the port when a TS-user issues a T-CONNECT.REQUEST action. This action specifies the TSAP ID of the remote TS-user, whether expedited data is to be supported, and (optionally) some initial TS-user data. The client consults the TSAP ID given to ascertain the IP address of the server. If the expedited data option was requested, the client opens a passive TCP port, in non-blocking mode, noting the port number. This TCP port is termed the "expedited port". The client then tries to open a TCP connection to the server on port 102. If not successful, the client fires T-DISCONNECT.INDICATION for the TS-user specifying the reason for failure (and, closes the expedited port, if any). If successful, the client sends a TPKT with code CR containing:

- the TSAP ID of the TS-user on the client's host (the "caller")
- the TSAP ID of the TS-user that the client wants to talk to (the "called")
- if the expedited data option was requested, the TSAP ID of the expedited port for the client's host
- any TS-user data from the T-CONNECT.REQUEST

The client now awaits a response.

The server, upon receipt of the TPKT, validates the contents of the TPKT (checking the version number, verifying that the code is CR, and so forth). If the packet is invalid, the server sends a TPKT with code DR specifying "PROTOCOL ERROR", closes the TCP connection, and goes back to the LISTEN state.

If the packet is valid, the server examines the TSAP ID that the remote TS-user wants to communicate with. If the TS-user specified can be located and started (e.g., the appropriate program which implements the indicated protocol is present), then the server starts this TS-user by firing T-CONNECT.INDICATION. Otherwise, the server sends a TPKT with code DR specifying "SESSION ENTITY NOT ATTACHED TO TSAP" or "REMOTE TRANSPORT ENTITY CONGESTED AT CONNECT REQUEST TIME" as appropriate, closes the TCP connection, and goes back to the LISTEN state.

The server now waits for a T-CONNECT.RESPONSE or T-DISCONNECT.REQUEST from the TS-user it started. if the latter is given, the server sends a TPKT with code DR containing the reason for the disconnect as supplied by the TS-user.

The server then closes the TCP connection and goes back to the LISTEN state.

Instead, if T-CONNECT.RESPONSE is given, the server sees if an expedited port was specified in the connection request. If so, the server opens a second TCP connection and connects to the specified port. If the connection fails, the server sends a TPKT with code DR specifying "CONNECTION NEGOTIATION FAILED", closes the TCP connection, and goes back to the LISTEN state. If the connection succeeded, the server notes the local port number used to connect to the expedited port.

If an expedited port was not specified in the TPKT with code CR, and the server's TS-user indicates that it wants to use expedited data, then the server sends a TPKT with code DR specifying "CONNECTION NEGOTIATION FAILED", fires T-DISCONNECT.INDICATION with this error to the TS-user, closes the TCP connection, and goes back to the LISTEN state.

The server now sends a TPKT with code CC containing:

- the TSAP ID of the TS-user responding to the connection (usually the "called")
- if an expedited port was specified in the TPKT with code CR,

the TSAP ID of the port number on the server's host that was used to connect to the expedited port
- any TS-user data from the T-CONNECT.RESPONSE

After sending the TPKT, the server enters the SYMMETRIC PEER state.

The client, upon receipt of the TPKT, validates the contents of the TPKT (checking the version number, verifying that the code is CC or DR, and so forth). If the packet is invalid, the client sends a TPKT with code DR specifying "PROTOCOL ERROR", fires T-DISCONNECT.INDICATION with this error to the TS-user, and closes the TCP connection (and the expedited port, if any).

If the packet's code is DR, the client fires T-DISCONNECT.INDICATION with the reason given in the TPKT to the TS-user, and closes the TCP connection (and the expedited port, if any).

If the packet's code is CC, the client checks if an expedited port was specified and that a connection is waiting on the expedited port. If not, a protocol error has occurred, a TPKT with code DR is returned, T-DISCONNECT.INDICATION is fired, and so on. Otherwise, the client checks the remote address that connected to the expedited port. If it differs from the port listed in the TPKT with code CC, a protocol error has occurred. Otherwise, all is well, two TCP connections have been established, one for all TPKTs except expedited data, and the second for the exclusive use of expedited data.

The client now fires T-CONNECT.CONFIRMATION, and enters the SYMMETRIC PEER state.

Once both sides have reached the SYMMETRIC PEER state, the protocol is completely symmetric, the notion of client/server is lost. Both TS-peers act in the following fashion:

If the TCP indicates that data can be read, the TS-peer, upon receipt of the TPKT, validates the contents. If the packet is invalid, the TS-peer sends a TPKT with code DR specifying "PROTOCOL ERROR", fires T-DISCONNECT.INDICATION with this error to the TS-user, and closes the TCP connection (and expedited data connection, if any). If the TS-peer was the server, it goes back to the LISTEN state.

NOTE: If the expedited data option was requested, then there are two TCP connections that can supply data for reading. The dialogue below assumes that only ED TPKTs are read from the expedited data connection. For simplicity's sake, when reading from TCP the relation between connections and TPKTs is unimportant and this memo URGES all implementations to be very lenient in this

regard. When writing to TCP, implementations should use the expedited data connection only to send TPKTs with code ED. Section 7 of this memo discusses the handling of expedited data in greater detail.

If the packet's code is DR, the TS-peer fires T-DISCONNECT.INDICATION with the reason given in the TPKT to the TS-user, and closes the TCP connection (and expedited data connection, if any). If the TS-peer was the server, it goes back to the LISTEN state.

If the packet's code is ED or DT, the TS-peer fires T-DATA.INDICATION or T-EXPEDITED DATA.INDICATION as appropriate with the enclosed user data for the TS-user. It then goes back to the SYMMETRIC PEER state.

If the packet is invalid, the TS-peer sends a TPKT with code DR specifying "PROTOCOL ERROR", fires T-DISCONNECT.INDICATION with this error to the TS-user, and closes the TCP connection (and expedited data connection, if any). If the TS-peer was the server, it goes back to the LISTEN state.

If the TCP indicates that an error has occurred and the connection has closed, then the TS-peer fires T-DISCONNECT.INDICATION to the TS-user specifying the reason for the failure. If the expedited data connection, if any, is still open, it is closed. If the TS-peer was the server, it goes back to the LISTEN state.

If the TS-user issues a T-DATA.REQUEST or T-EXPEDITED DATA.REQUEST action, the TS-peer sends a TPKT with code DT or ED containing the TS-user data. It then goes back to the SYMMETRIC PEER state.

If the TS-user issues a T-DISCONNECT.REQUEST action, the TS-peer sends a TPKT with code DR containing the reason for the disconnect as supplied by the TS-user. The TS-peer then closes the TCP connection, (and expedited data connection, if any). If the TS-peer was the server, it goes back to the LISTEN state.

In terms of (augmented) state tables, the protocol can be explained as follows. The server starts in state S0, the client starts in state C0. "TCP:" refers to an event or action from the TCP service, "SS:" refers to an event or action from the TS-user (e.g., the ISO session service [ISO-8327]).

S E R V E R S T A T E S

state	event	action	goto
S0		TCP: listen on port 102	S1
S1	TCP: connected	TCP: read TPKT	
		parse, on error	
		TCP: send DR, close	S0
		code is CR	
		start session server	
		SS: T-CONNECT .INDICATION	S2
		otherwise,	
		TCP: send DR, close	S0
S2	SS: T-CONNECT.RESPONSE	if expedited option,	
		TCP: open port EXPD	
		TCP: send CC	P0
S2	SS: T-DISCONNECT .REQUEST	TCP: send DR, close	S0

Any event occuring for a state not listed above is considered an error, and handled thusly:

state	event	action	goto
S*	TCP: other	if TCP is open, TCP: close	S0
		otherwise ignore	S0
S*	SS: other	SS: T-DISCONNECT .INDICATION	
		if TCP is open, close	S0

C L I E N T S T A T E S

state	event	action	goto

C0	SS: T-CONNECT.REQUEST	if expedited option,
		  TCP: non-blocking
		  listen on port EXPD
		TCP: open port 102
			
			C1

C1	TCP: connected	TCP: send CR	C2
C1	TCP: connect fails	TCP: close SS: T-DISCONNECT .INDICATION	 C0

C2	TCP: data ready	TCP: read TPKT
		parse, on error
		  TCP: send DR, close
		  SS: T-DISCONNECT
		  .INDICATION
		code is CC
		  if expedited option,
		  verify port EXPD
		  connected correctly,
		  if not, treat as error
		  SS: T-CONNECT
		  .CONFIRMATION
		code is DR
		  TCP: close
		  SS: T-DISCONNECT
		  .INDICATION
		otherwise
		  TCP: send DR, close
		  SS: T-DISCONNECT
		  .INDICATION
			
			C0
			
			P0
			
			C0
			
			C0

Any event occuring for a state not listed above is considered an error, and handled thusly:

state	event	action	goto
C*	TCP: other	if TCP is open, close	C0
		otherwise ignore	C0
C*	SS: other	SS: T-DISCONNECT .INDICATION	
		if TCP is open, close	C0

P E E R S T A T E S

state	event	action	goto
P0	TCP: data ready	TCP: read TPKT	
		parse, on error	
		TCP: send DR, close	
		SS: T-DISCONNECT .INDICATION	end
		code is DT	
		SS: T-DATA.INDICATION	P0
		code is ED	
		SS: T-EXPEDITED DATA .INDICATION	P0
		code is DR	
		TCP: close	
		SS: T-DISCONNECT .INDICATION	end
		otherwise	
		TCP: send DR, close	
		SS: T-DISCONNECT .INDICATION	end
P0	TCP: other	TCP: close	
		SS: T-DISCONNECT .INDICATION	end
P0	SS: T-DATA.REQUEST	TCP: send DT	P0
P0	SS: T-EXPEDITED DATA .REQUEST	TCP: send ED	P0
P0	SS: T-DISCONNECT .REQUEST	TCP: send DR, close	end
P0	SS: other	TCP: send DR, close	
		SS: T-DISCONNECT .INDICATION	end

6. Packet Format

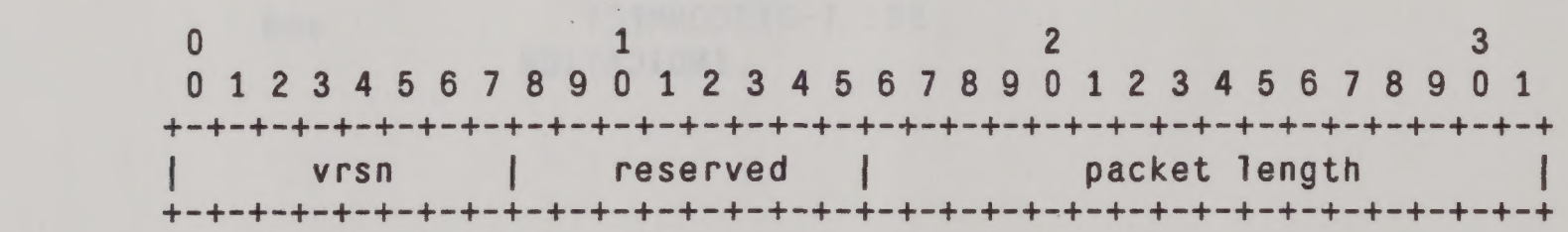
Two TS-peers exchange information over a TCP connection by encapsulating information in well-defined packets. A packet, denoted as "TPKT" in the previous sections, is viewed as an object composed of an integral number of octets, of variable length.

> NOTE: For the purposes of presentation, these objects are shown as being 4 octets (32 bits wide). This representation is an artifact of the style of this memo and should not be interpreted as requiring that a TPDU be a multiple of 4 octets in length.

A packet consists of two parts: a packet-header and a pseudo-TPDU. The format of the header is constant regardless of the type of packet. The format of the pseudo-TPDU follows the [ISO-8073] recommendation very closely with the exceptions listed below. As per [ISO-8073], each TPDU consists of two parts: header and data.

It is EXTREMELY important to observe that TPKTs represent "indivisible" units of data to the TS-user. That is, a T-DATA.REQUEST initiated by the TS-user at the sending end of a connection should result in exactly one T-DATA.INDICATION being generated (with exactly that data) for the TS-user at the receiving end. To ensure this behavior, it is critical that any INDICATION event resulting from a TPKT be initiated ONLY after the entire TPKT is fully received. Furthermore, exactly one such INDICATION event should be generated by the TS-peer. The packet length field, as described below, can accommodate on the order of 65K octets of user data. This should be well above the requirements of the size of any SPDU (Session Protocol Data Unit) for any real implementation. As a result, version 1 of this protocol has no need to either fragment or re-assemble TS-user data. If an application arises which requires an SPDU of size greater than 65K octets, this memo will be revised.

The format of the packet-header is as follows:



where:

1. vrsn — 8 bits

   This field is always 1 for this memo.

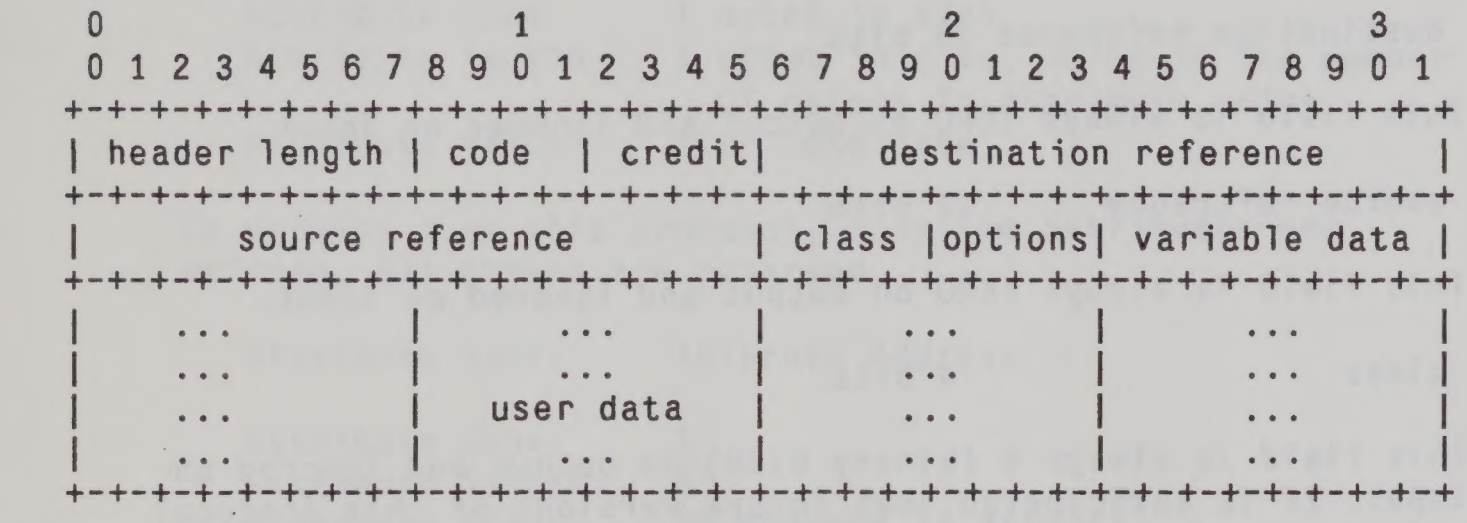
2. packet length 16 bits (min=8, max=65535)

The length of entire packet in octets, including packet-header.

The format of the TPDU (to re-phrase from [ISO-8073]) depends on the type of a TPDU. All TPDUs start with a fixed-part header length and the code. The information following after the code varies, depending on the value of the code. In the context of this memo, the following codes are valid:

CR: connect request
CC: connect confirm
DR: disconnect request
DT: data
ED: expedited data

The format of a CR or CC TPDU is:


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
header length
code
credit
destination reference
source reference
class
options
variable data
...
user data

where:

3. header length 8 bits (min=6, max=min(254,packet length-6))

The TPDU-header length in octets including parameters but excluding the header length field and user data (if any).

4. code 4 bits

The type of TPDU. Values, in the context of this memo, are:

value	meaning	
14	CR: connection request	(binary 1110)
13	CC: connection confirm	(binary 1101)
8	DR: disconnect request	(binary 1000)
15	DT: data	(binary 1111)
1	ED: expedited data	(binary 0001)
all other	reserved	

5. credit 4 bits

This field is always ZERO on output and ignored on input.

6. destination reference 16 bits

This field is always ZERO on output and ignored on input.

7. source reference 16 bits

This field is always ZERO on output and ignored on input.

8. class 4 bits

This field is always 4 (binary 0100) on output and ignored on input. It is anticipated that future versions of this protocol will make use of this field.

9. options 4 bits

This field is always ZERO on output and ignored on input.

10. variable data (header length - 6) octets

This portion of the TPDU is of variable length. For most TPDUs, this portion is empty (the header length field of the TPDU is exactly 6). The contents of the variable data consist of zero or more "parameters". Each parameter has the following format:

parameter code	1 octet in size
parameter length	1 octet in size, value is the number of octets in parameter value
parameter value	parameter data

Normally, the parameter length is 1 octet. Any implementation conforming to this version of the protocol must recognize all parameter types listed in [ISO-8073]. With the exception of the parameters listed below, these parameters are simply ignored.

o Parameter name: Transport service access point identifier (TSAP-ID) of the client TSAP

Parameter code:	193 (binary 1100 0001)
Parameter length:	variable
Parameter value:	TSAP-ID attributes

Each TSAP-ID consists of 1 or more attributes. Each attribute has this format:

Attribute code	1 octet in size
Attribute length	1 octet in size, value is the number of octets in attribute value
Attribute value	attribute data

In version 1 of this protocol, only two attributes are defined. All others are reserved.

Attribute name:	Internet Address
Attribute code:	1
Attribute length:	6
Attribute value:	IP address (4 octets) session port (2 octets, unsigned integer)

This attribute is ALWAYS required. Values for session port can be found in Appendix A of this memo.

Attribute name:	Internet Address for Expedited Data
Attribute code:	2
Attribute length:	6
Attribute value:	IP address (4 octets) TCP port (2 octets, unsigned integer)

This attribute is required ONLY if expedited data is to be exchanged. The attribute value is an <IP address, TCP port> pair designated by the TS-peer for use with expedited data.

o Parameter name: TSAP-ID of the server TSAP

Parameter code: 194 (binary 1100 0010)
Parameter length: variable
Parameter value: TSAP-ID attributes

o Parameter name: Additional option selection

Parameter code: 198 (binary 1100 0110)
Parameter length: 1
Parameter value: additional flags

The additional flags octet consists of 8-bits of optional flags. Only one bit is of interest to this memo, the remaining bits should be ZERO on output and ignored on input. This bit indicates if the transport expedited data service is to be used. If this bit is set (bit mask 0000 0001) or this parameter does not appear in the TPDU, then the expedited data service is requested. If this parameter appears in the TPDU and the bit is not set then the service is disabled. If the service is requested, then the TSAP-ID of the sender of the TPDU must include an attribute indicating the internet address to use for expedited data.

o Parameter name: Alternative protocol classes

Parameter code: 199 (binary 1100 0111)

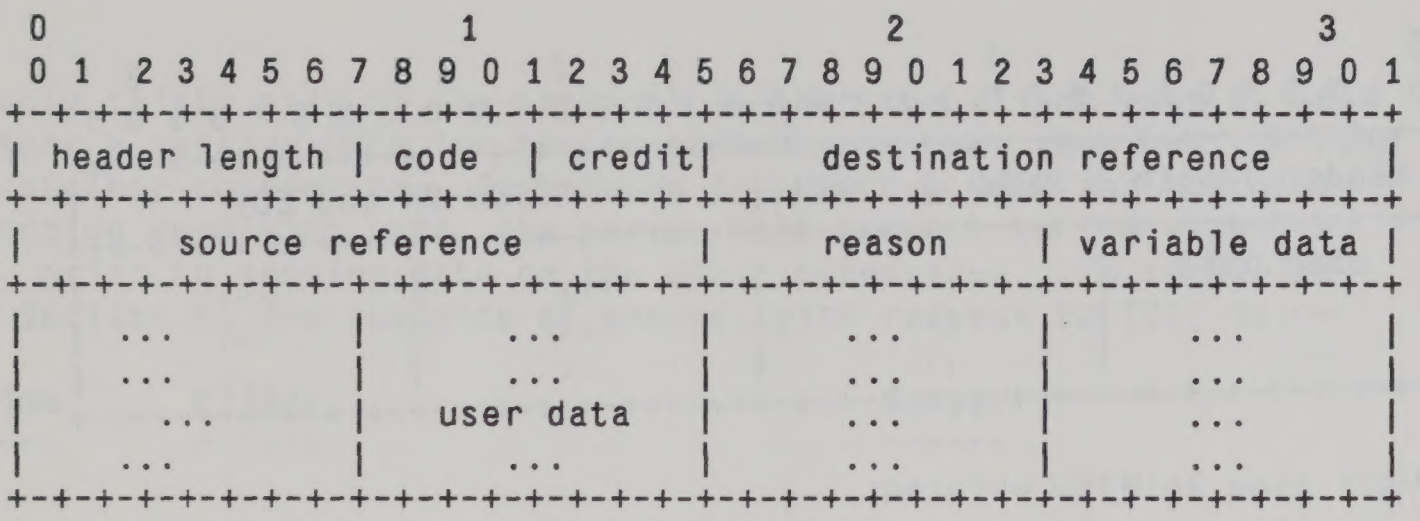
Parameter length: variable
Parameter value: 64 (binary 0100 0000) in each octet

This is used as a NOOP in the variable data. Its use is HIGHLY discouraged, but for those implementors who wish to align the user data portion of the TPDU on word (or page) boundaries, use of this parameter for filling is recommended.

11. user data (packet length - header length - 5) octets

This portion of the TPDU is actual user data, most probably one or more SPDUs (session protocol data units).

The format of a DR TPDU is:


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
header length
code
credit
destination reference
source reference
reason
variable data
...
user data

The format of the fields is identical to those of a CR or CC TPDU, with the following exceptions:

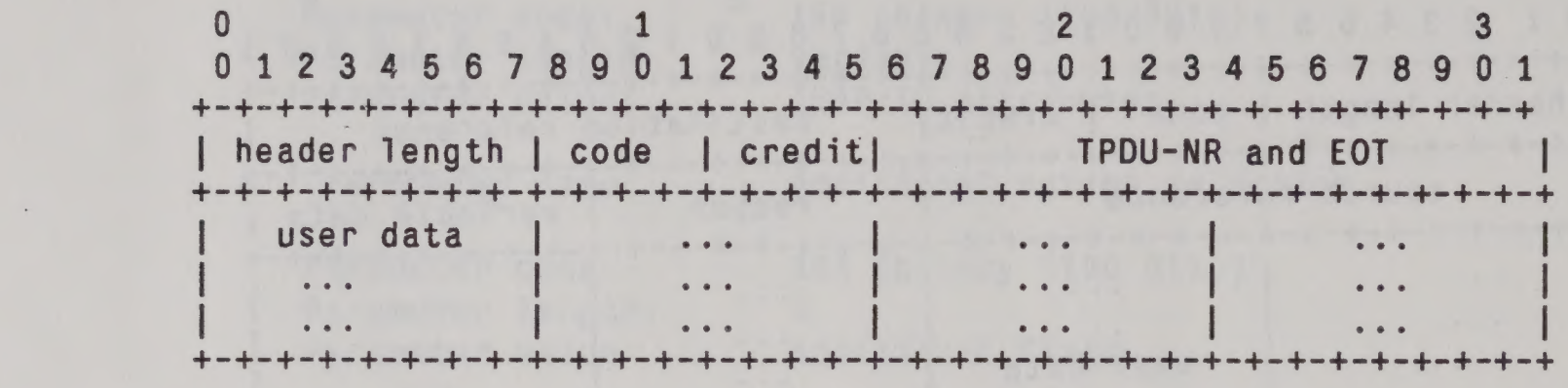
where:

8. reason 8 bits

This replaces the class/option fields of the CR or CC TPDU. Any value, as specified in [ISO-8073], may be used in this field. This memo makes use of several:

value	meaning
1	Congestion at TSAP
2	Session entity not attached to TSAP
3	Address unknown (at TCP connect time)
128+0	Normal disconnect initiated by the session entity
128+1	Remote transport entity congestion at connect request time
128+3	Connection negotiation failed
128+5	Protocol Error
128+8	Connection request refused on this network connection

The format of a DT or ED TPDU is:



where:

After the credit field (which is always ZERO on output and ignored on input), there is one additional field prior to the user data:

6. TPDU-NR and EOT 16 bits

This field is always ZERO on output and ignored on input.

7. Expedited Data

This memo utilizes a second TCP connection to handle expedited data and does not make use of the TCP URGENT mechanism. The primary disadvantage of this decision is that single-threaded implementations of TCP may have some difficulty in supporting two simultaneous connections. There are however several advantages to this approach:

a. Use of a single connection to implement the semantics of expedited data implies that the TSAP peer manage a set of buffers independent from TCP. The peer would, upon indication of TCP urgent information, have to buffer all preceeding TPKTs until the TCP buffer was empty. Expedited data would then be given to the TS-user. When the expedited data was flushed, then the buffered (non-expedited) data could be passed up to the receiving user.

b. It assumes that implementations support TCP urgency correctly. This is perhaps an untrue assumption, particular in the case of TCP urgency occuring when the send window is zero-sized. Further, it assumes that the implementations can signal this event to the TCP-user in a meaningful fashion. In a single-threaded implementation, this is not likely.

Given a reasonable TCP implementation, the TS-peer need listen on two

TCP sockets in either polling or interrupt mode. When the TS-peer is given data, the TCP must indicate which connection should be read from.

The only tricky part of the protocol is that the client must be able to start a passive OPEN for the expedited port, and then wait to read from another connection. In between the passive OPEN and the other connection supplying data, the server will connect to the expedited port, prior to sending data on the other connection. To summarize from Section 5, the sequence of events, with respect to TCP, is:

time	client	Server
0.		passive OPEN of port 102
1.	T-CONNECT.REQUEST from user passive OPEN of expedited port (non-blocking)	
2.	active OPEN of port 102	
3.	send CC TPKT	
4.		port 102 connected
5.		receive CC TPKT T-CONNECT.INDICATION to user T-CONNECT.RESPONSE from user
6.		active OPEN to expedited port
7.	expedited port connected	
8.		send CR TPKT
9.	receive CR TPKT verify expedited port connected correctly	

Multi-threaded implementations of TCP should be able to support this sequence of events without any great difficulty.

8. Conclusions

There are two design decisions which should be considered. The first deals with particular packet format used. It should be obvious to the reader that the TP packet format was adopted for use in this memo. Although this results in a few fields being ignored (e.g., source reference), it does not introduce an unacceptable amount of overhead. For example, on a connection request packet (the worst case) there are 6 bytes of "zero on output, ignore on input" fields. Considering that the packet overhead processing is fixed, requiring that implementations allocate an additional 1.5 words is not unreasonable! Of course, it should be noted that some of these fields (i.e., class) may be used in future versions of the protocol as experience is gained.

The second decision deals with how the TCP and TSAP port space is administered. It is probably a very bad idea to take any responsibility, whatsoever, for managing this addressing space, even after ISO has stabilized. There are two issues involved. First, at what level do the TCP and TSAP port spaces interact; second, who defines what this interaction looks like. With respect to the first, it wholly undesirable to require that each TSAP port map to a unique TCP port. The administrative problems for the TCP "numbers czar (and czarina)" would be non-trivial. Therefore, it is desirable to allocate a single TCP port, namely port 102, as the port where the "ISO Transport Services" live in the TCP domain. Second, the interaction defined in Appendix A of this memo is embryonic at best. It will no doubt be replaced as soon as the ISO world reaches convergence on how services are addressed in ISO TP. Therefore readers (and implementors) are asked to keep in mind that this aspect of the memo is guaranteed to change. Unfortunately, the authors are not permitted the luxury of waiting for a consensus in ISO. As a result, the minimal effort approach outlined in the appendix below was adopted.

A prototype implementation of the protocol described by this memo is available for 4.2BSD UNIX. Interested parties should contact the authors for a copy. To briefly mention its implementation, a given ISO service is implemented as a separate program. A daemon listens on TCP port 102, consults a database when a connection request packet is received, and fires the appropriate program for the ISO service requested. Of course, given the nature of the BSD implementation of TCP, as the child fires, responsibility of that particular connection is delegated to the child; the daemon returns to listening for new connection requests. The prototype implementation consists of both the daemon program and subroutine libraries which are loaded with programs providing ISO services.

9. References

[ISO-8072] ISO.
"International Standard 8072. Information Processing Systems -- Open Systems Interconnection: Transport Service Definition."
(June, 1984)

[ISO-8073] ISO.
"International Standard 8073. Information Processing Systems -- Open Systems Interconnection: Transport Protocol Specification."
(June, 1984)

[ISO-8327] ISO.
"International Standard 8327. Information Processing Systems -- Open Systems Interconnection: Session Protocol Specification."
(June, 1984)

[RFC-791] Internet Protocol.
Request for Comments 791
(September, 1981)
(See also: MIL-STD-1777)

[RFC-793] Transmission Control Protocol.
Request for Comments 793
(September, 1981)
(See also: MIL-STD-1778)

[RFC-960] Assigned Numbers.
Request for Comments 960
(December, 1985)

[X.214] CCITT.
"Recommendation X.214. Transport Service Definitions for Open Systems Interconnection (OSI) for CCITT Applications."
(October, 1984)

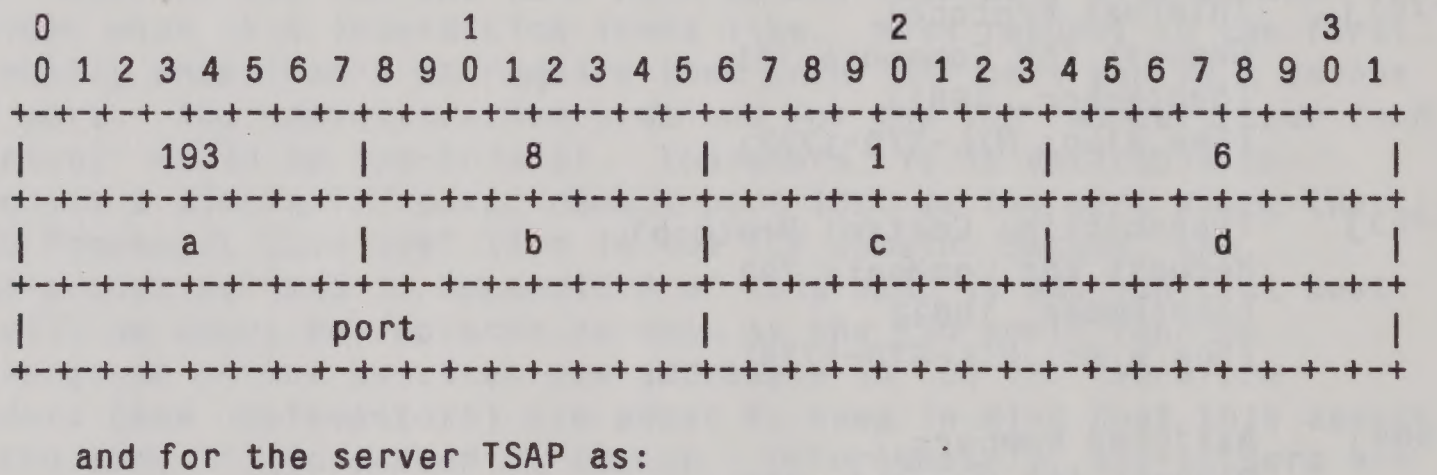
[X.224] CCITT.
"Recommendation X.224. Transport Protocol Specification for Open Systems Interconnection (OSI) for CCITT Applications."
(October, 1984)

[X.225] CCITT.
"Recommendation X.225. Session Protocol Specification for Open Systems Interconnection (OSI) for CCITT Applications."
(October, 1984)

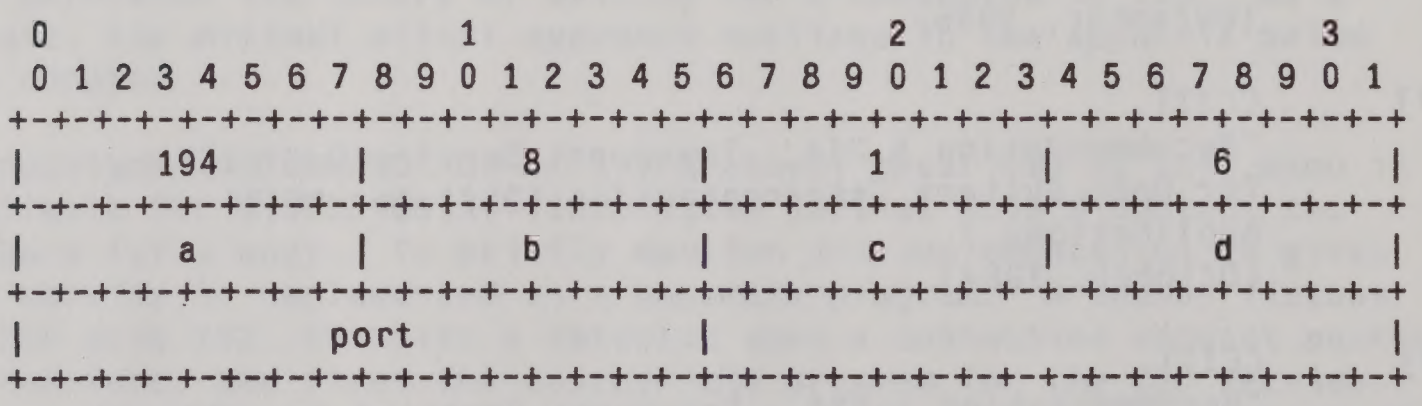
[X.410] CCITT.
"Recommendation X.410. Message Handling Systems: Remote Operations and Reliable Transfer Server."
(October, 1984)

Appendix A: Reserved TSAP IDs

Version 1 of this protocol uses a relatively simple encoding scheme for TSAP IDs. A TSAP ID is an attribute list containing two parameters, a 32-bit IP address, and a 16-bit port number. This is used to identify both the client TSAP and the server TSAP. When it appears in a TPKT with code CR or CC, the TSAP ID is encoded in the variable data part for the client TSAP as:



and for the server TSAP as:



(Neither of these examples include an attribute for a TCP connection for expedited data. If one were present, the length of the TSAP ID attribute would be 16 instead of 8, and the 8 bytes following the Internet address would be "2" for the attribute code, "6" for the

attribute length, and then 6 octets for the Internet address to use for expedited data, 4 octets for IP address, and 2 octets for TCP port.)

Where [a.b.c.d] is the IP address of the host where the respective TSAP peer resides, and port is a 16-bit unsigned integer describing where in the TSAP port space the TS-user lives.

Port value	Designation
0	illegal
1-4096	privileged
4097-65535	user

The following table contains the list of the "official" TSAP ID port numbers as of the first release of this memo. It is expected that future editions of the "Assigned Numbers" document[RFC-960] will contain updates to this list.

Port	name	ISO service
1	echo	unofficial echo
2	sink	unofficial data sink
3	FTAM	File Transfer, Access, and Management
4	VTS	ISO-8571 Virtual Terminal Service
5	MHS	Message Handling System [X.411] CCITT X.400
6	JTM	Job Transfer and Manipulation ISO 8831/8832
7	CASE	Common Application Service Elements Kernel ISO-8650/2

If anyone knows of a list of "official" ISO services, the authors would be very interested.